Proof-Relevant Resolution for Constructive Automation

František Farka

April 29, 2019

Heriot-Watt University and University of St Andrews

1/10 - 4 min / 4 min total
Susped screensaver, . . .

1. thx, plsr
2. about – what is the context
3. automation in prg languages
4. safety, security, verification via types
5. let us have examples

Automation for Programming Languages

Automation for Programming Languages

Type inference, term synthesis, and type classes

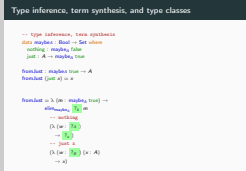
```
-- type inference, term synthesis
data maybe_A : Bool → Set where
  nothing : maybe_A false
  just : A → maybe_A true

fromJust : maybe_A true → A
fromJust (just x) = x
```


2/10 - 8 min / 12 min total

1. maybe haskell, option ocaml, fromjust partial
2. dependent types to rescue
3. technical solution

Type inference, term synthesis, and type classes

```
-- type inference, term synthesis
data maybe_A : Bool → Set where
  nothing : maybe_A false
  just : A → maybe_A true

fromJust : maybe_A true → A
fromJust (just x) = x


fromJust = λ (m : maybe_A true) →
        elim_maybe_A ?_b m
          -- nothing
          (λ (w : ?_A )
            → ?_e )
          -- just x
          (λ (w : ?_B ) (x : A)
            → x)
```

2/10 - 8 min / 12 min total

1. maybe haskell, option ocaml, fromjust partial
2. dependent types to rescue
3. technical solution
4. internal representation
5. automation to fill missing

Type inference, term synthesis, and type classes

```
-- type inference, term synthesis
data maybe_A : Bool → Set where
  nothing : maybe_A false
  just : A → maybe_A true

fromJust : maybe_A true → A
fromJust (just x) = x


fromJust = λ (m : maybe_A true) →
         elim_maybe_A true m
           -- nothing
           (λ (w : true ≡ false )
             → elim_≡ w )
           -- just x
           (λ (w : true ≡ true ) (x : A)
             → x)
```

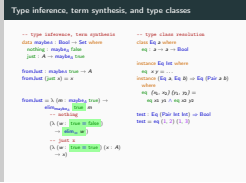

2/10 - 8 min / 12 min total

1. lets have a look at another example
2. type classes in Haskell
3. again internal representation

Type inference, term synthesis, and type classes

```
-- type inference, term synthesis
data maybe_A : Bool → Set where
  nothing : maybe_A false
  just : A → maybe_A true

fromJust : maybe_A true → A
fromJust (just x) = x


fromJust = λ (m : maybe_A true) →
        elim_maybe_A true m
          -- nothing
          (λ (w : true ≡ false )
            → elim_≡ w )
          -- just x
          (λ (w : true ≡ true ) (x : A)
            → x)
```

```
-- type class resolution
class Eq a where
  eq : a → a → Bool

instance Eq Int where
  eq  x y = ...
instance (Eq a, Eq b) ⇒ Eq (Pair a b)
  where
  eq  (x1, x2) (y1, y2) =
    eq x1 y1 ∧ eq x2 y2

test : Eq (Pair Int Int) ⇒ Bool
test = eq (1, 2) (1, 3)
```

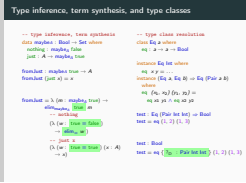
2/10 - 8 min / 12 min total

1. lets have a look at another example
2. type classes in Haskell
3. again internal representation

Type inference, term synthesis, and type classes

```
-- type inference, term synthesis
data maybe_A : Bool → Set where
  nothing : maybe_A false
  just : A → maybe_A true

fromJust : maybe_A true → A
fromJust (just x) = x


fromJust = λ (m : maybe_A true) →
        elim_maybe_A true m
          -- nothing
          (λ (w : true ≡ false )
            → elim≡ w )
          -- just x
          (λ (w : true ≡ true ) (x : A)
            → x)
```

```
-- type class resolution
class Eq a where
  eq : a → a → Bool

instance Eq Int where
  eq  x y = ...
instance (Eq a, Eq b) ⇒ Eq (Pair a b)
  where
  eq  (x1, x2) (y1, y2) =
    eq x1 y1 ∧ eq x2 y2

test : Eq (Pair Int Int) ⇒ Bool
test = eq (1, 2) (1, 3)


test : Bool
test = eq {?D : Pair Int Int} (1, 2) (1, 3)
```

2/10 - 8 min / 12 min total

1. lets have a look at another example
2. type classes in Haskell
3. again internal representation
4. DICTIONARY
5. this is the kind of problems
6. framework, general and principled
7. ppl argued for HC's and it works for TC's (citations)
8. comples, look at TC's

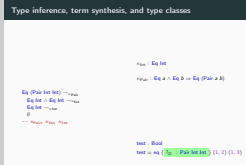
Type inference, term synthesis, and type classes

κ_{Int} : Eq Int

κ_{Pair} : Eq a ∧ Eq b ⇒ Eq (Pair a b)

Eq (Pair Int Int) $\rightsquigarrow_{\kappa_{\mathbf{Pair}}}$
  Eq Int ∧ Eq Int $\rightsquigarrow_{\kappa_{\mathbf{Int}}}$
  Eq Int $\rightsquigarrow_{\kappa_{\mathbf{Int}}}$
  $\emptyset$
-- κ_{Pair} κ_{Int} κ_{Int}

test : Bool
test = eq {?$_{\mathsf{D}}$: Pair Int Int} (1, 2) (1, 3)



3/10 - 4 min / 16 min total

1. typeclasses, names
2. goal - instance
3. resolution - unification, substitution
4. proof-term = dictionary

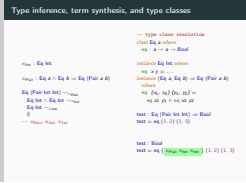
Type inference, term synthesis, and type classes

κ_{Int} : Eq Int

κ_{Pair} : Eq $a \wedge$ Eq $b \Rightarrow$ Eq (Pair a b)

Eq (Pair Int Int) $\rightsquigarrow_{\kappa_{\text{Pair}}}$
  Eq Int $\wedge$ Eq Int $\rightsquigarrow_{\kappa_{\text{Int}}}$
  Eq Int $\rightsquigarrow_{\kappa_{\text{Int}}}$
  $\emptyset$
-- κ_{Pair} κ_{Int} κ_{Int}

```
-- type class resolution
class Eq a where
  eq : a → a → Bool

instance Eq Int where
  eq  x y = ...
instance (Eq a, Eq b) ⇒ Eq (Pair a b)
  where
  eq  (x1, x2) (y1, y2) =
    eq x1 y1 ∧ eq x2 y2

test : Eq (Pair Int Int) ⇒ Bool
test = eq (1, 2) (1, 3)


test : Bool
test = eq {κPair κInt κInt} (1, 2) (1, 3)
```

3/10 - 4 min / 16 min total

1. similarly for TITS, complex, omit
2. proof relevance in TITS - later
3. we present the framework, HH flae though
4. inspired by LP, goal directed search

Proof-Relevant Resolution

Big-step operational semantics

$$D := A \mid G \Rightarrow D \mid \forall x : A.D$$
$$G := A \mid D \Rightarrow G \mid \forall x : A.G \mid \exists x : A.G$$
$$e := \kappa \mid e\ e \mid \lambda\kappa.e \mid \langle M, e\rangle$$

$$\boxed{\mathcal{S}; \mathcal{P} \longrightarrow e : G}$$

$$\boxed{\mathcal{S}; \mathcal{P} \xrightarrow{e':D} e : A}$$

$$\frac{}{\mathcal{S}; \mathcal{P} \xrightarrow{e:A} e : A}\ \text{init}$$

$$\frac{\mathcal{S}; \mathcal{P} \xrightarrow{\kappa:D} e : A \qquad \kappa : D \in \mathcal{P}}{\mathcal{S}; \mathcal{P} \longrightarrow e : A}\ \text{decide}$$

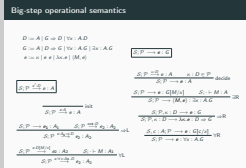
$$\frac{\mathcal{S}; \mathcal{P} \longrightarrow \mathbf{e_1} : A_1 \qquad \mathcal{S}; \mathcal{P} \xrightarrow{e\mathbf{e_1}:D} e_2 : A_2}{\mathcal{S}; \mathcal{P} \xrightarrow{e:\mathbf{A_1}\Rightarrow D} e_2 : A_2}\ \Rightarrow\text{L}$$

$$\frac{\mathcal{S}; \mathcal{P} \xrightarrow{e:D[M/x]} e_2 : A_2 \qquad \mathcal{S}; \cdot \vdash M : A_1}{\mathcal{S}; \mathcal{P} \xrightarrow{e:\forall x:\mathbf{A_1}.D} e_2 : A_2}\ \forall\text{L}$$

$$\frac{\mathcal{S}; \mathcal{P} \longrightarrow e : G[M/x] \qquad \mathcal{S}; \cdot \vdash M : A}{\mathcal{S}; \mathcal{P} \longrightarrow \langle M, e\rangle : \exists x : A.G}\ \exists\text{R}$$

$$\frac{\mathcal{S}; \mathcal{P}, \kappa : D \longrightarrow e : G}{\mathcal{S}; \mathcal{P}, \kappa : D \longrightarrow \lambda\kappa.e : D \Rightarrow G}\ \Rightarrow\text{R}$$

$$\frac{\mathcal{S}, c : A; \mathcal{P} \longrightarrow e : G[c/x]}{\mathcal{S}; \mathcal{P} \longrightarrow e : \forall x : A.G}\ \forall\text{R}$$



4/10 - 4 min / 20 min total

1. language, D, G, reversed roles, e
2. atoms - LF, expressive, not important for the rest
3. semantics - uniform proofs
4. exaplain wia LJ, righ goal directed, left restrics LJ to resolution
5. in resolution - unif, subst, not here - no computational device
6. exists - term, init - equal
7. next small

Small-step operational semantics

$$\hat{e} := \kappa \mid G \mid \hat{e}\,\hat{e} \mid \langle M, \hat{e}\rangle \mid \lambda\kappa.\hat{e}$$
$$C := \bullet \mid e\ C \mid \langle M, C\rangle \mid \lambda\kappa.C$$

$$\boxed{\mathcal{S};\mathcal{P} \vdash \Gamma \mid \hat{e} \overset{\hat{e}'':D}{\rightsquigarrow} \Gamma' \mid \hat{e}'}$$

$$\frac{\mathcal{S};\Gamma \vdash \sigma : \Gamma' \qquad \mathcal{S};\Gamma' \vdash \sigma A \equiv \sigma A' : \mathsf{o}}{\mathcal{S};\mathcal{P} \vdash \Gamma \mid C\{A\} \overset{\hat{e}:A'}{\rightsquigarrow} \Gamma' \mid (\sigma C)\{\hat{e}\}}$$

$$\frac{\mathcal{S};\mathcal{P} \vdash \Gamma \mid C\{A\} \overset{\hat{e}_1\ A_1:D}{\rightsquigarrow} \Gamma' \mid \hat{e}}{\mathcal{S};\mathcal{P} \vdash \Gamma \mid C\{A\} \overset{\hat{e}_1:A_1 \Rightarrow D}{\rightsquigarrow} \Gamma' \mid \hat{e}}$$

$$\frac{\mathcal{S};\mathcal{P} \vdash \Gamma, Y : A_1 \mid C\{A_2\} \overset{\hat{e}_1:D[Y/x]}{\rightsquigarrow} \Gamma' \mid \hat{e}}{\mathcal{S};\mathcal{P} \vdash \Gamma \mid C\{A_2\} \overset{\hat{e}_1:\forall x:A_1.D}{\rightsquigarrow} \Gamma' \mid \hat{e}}$$

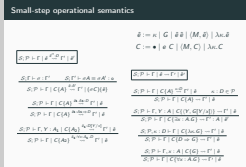
$$\boxed{\mathcal{S};\mathcal{P} \vdash \Gamma \mid \hat{e} \rightsquigarrow \Gamma' \mid \hat{e}'}$$

$$\frac{\mathcal{S};\mathcal{P} \vdash \Gamma \mid C\{A\} \overset{\kappa:D}{\rightsquigarrow} \Gamma' \mid \hat{e} \qquad \kappa : D \in \mathcal{P}}{\mathcal{S};\mathcal{P} \vdash \Gamma \mid C\{A\} \rightsquigarrow \Gamma' \mid \hat{e}}$$

$$\frac{\mathcal{S};\mathcal{P} \vdash \Gamma, Y : A \mid C\{\langle Y, G[Y/x]\rangle\} \rightsquigarrow \Gamma' \mid \hat{e}}{\mathcal{S};\mathcal{P} \vdash \Gamma \mid C\{\exists x : A.G\} \rightsquigarrow \Gamma' : A \mid \hat{e}'}$$

$$\frac{\mathcal{S};\mathcal{P}, \kappa : D \vdash \Gamma \mid C\{\lambda\kappa.G\} \rightsquigarrow \Gamma' \mid \hat{e}}{\mathcal{S};\mathcal{P} \vdash \Gamma \mid C\{D \Rightarrow G\} \rightsquigarrow \Gamma' \mid \hat{e}}$$

$$\frac{\mathcal{S};\mathcal{P} \vdash \Gamma, x : A \mid C\{G\} \rightsquigarrow \Gamma' \mid \hat{e}}{\mathcal{S};\mathcal{P} \vdash \Gamma \mid C\{\forall x : A.G\} \rightsquigarrow \Gamma' \mid \hat{e}}$$



5/10 - 4 min / 20 min total

1. not going to detail, corresponds to big
2. Big on proof terms, here intermediate mixed, contexts
3. init - replaced by wf. subst
4. LP - recover answer subst, similarly for impl quant
5. Question: Is it sound?

Soundness

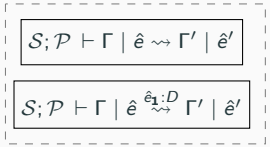
Theorem (Soundness)
If $\mathcal{S}; \mathcal{P} \vdash \cdot \mid G \rightsquigarrow \cdot \mid e$ *then* $\mathcal{S}; \mathcal{P} \longrightarrow e : G$.

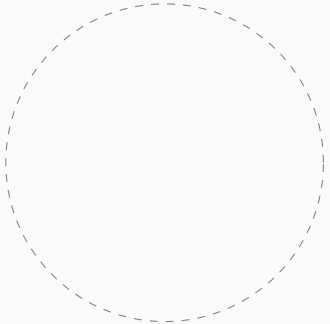
Theorem (Generalised soundness)
If $\mathcal{S}; \mathcal{P} \vdash \cdot \mid G \rightsquigarrow \Gamma' \mid e$ *then* $\mathcal{S}; \mathcal{P} \longrightarrow e : \forall \Gamma'.G$.

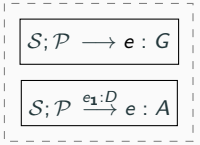
Soundness
Theorem (Soundness)
If $\mathcal{S}; \mathcal{P} \vdash \cdot \mid G \rightsquigarrow \cdot \mid e$ then $\mathcal{S}; \mathcal{P} \longrightarrow e : G$.
Theorem (Generalised soundness)
If $\mathcal{S}; \mathcal{P} \vdash \cdot \mid G \rightsquigarrow \Gamma' \mid e$ then $\mathcal{S}; \mathcal{P} \longrightarrow e : \forall \Gamma'.G$.

6/10 - 2 min / 22 min total

1. Yes, it is
2. Moreover, we can generalise

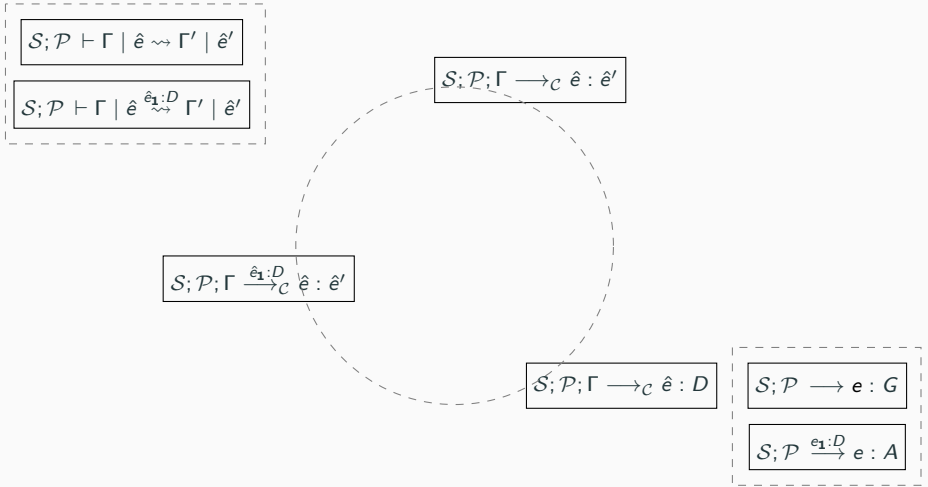


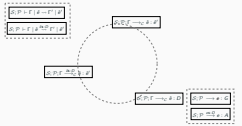




7/10 - 8 min / 30 min total

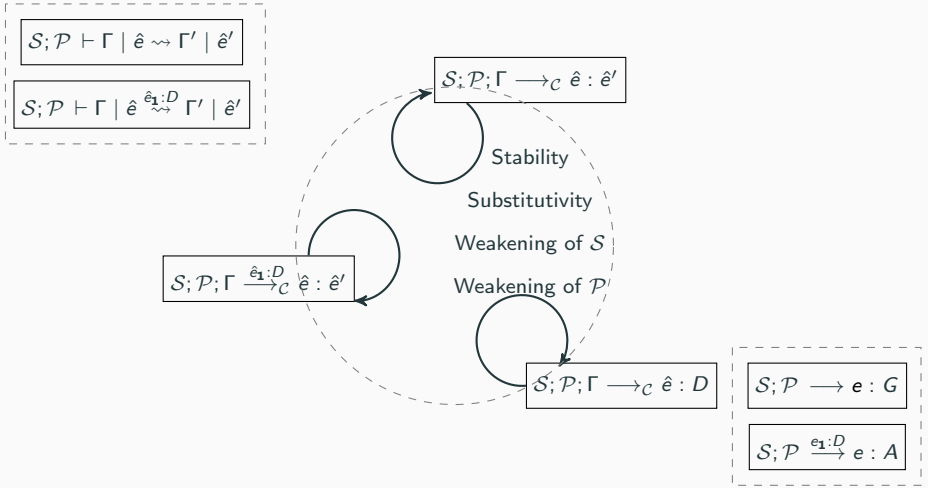
1. Yes, it is
2. proof outline
3. from Small to Big, middle missing. Issue: proof vs mixed, unification vars vs ground
4. we devise logical relation

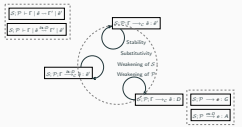




7/10 - 8 min / 30 min total

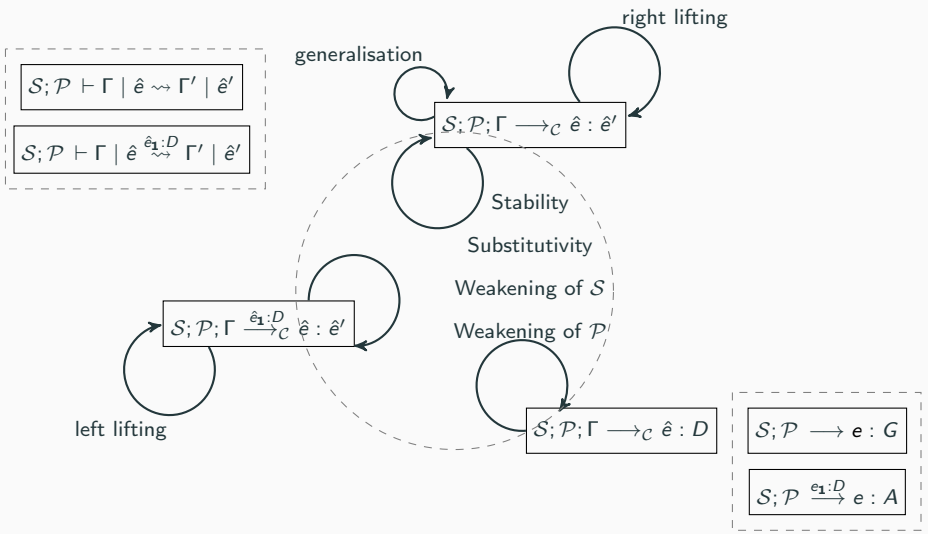
1. Yes, it is
2. proof outline
3. from Small to Big, middle missing. Issue: proof vs mixed, unification vars vs ground
4. we devise logical relation
5. have structural properties

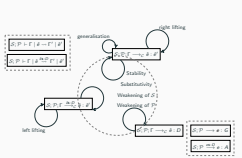




7/10 - 8 min / 30 min total

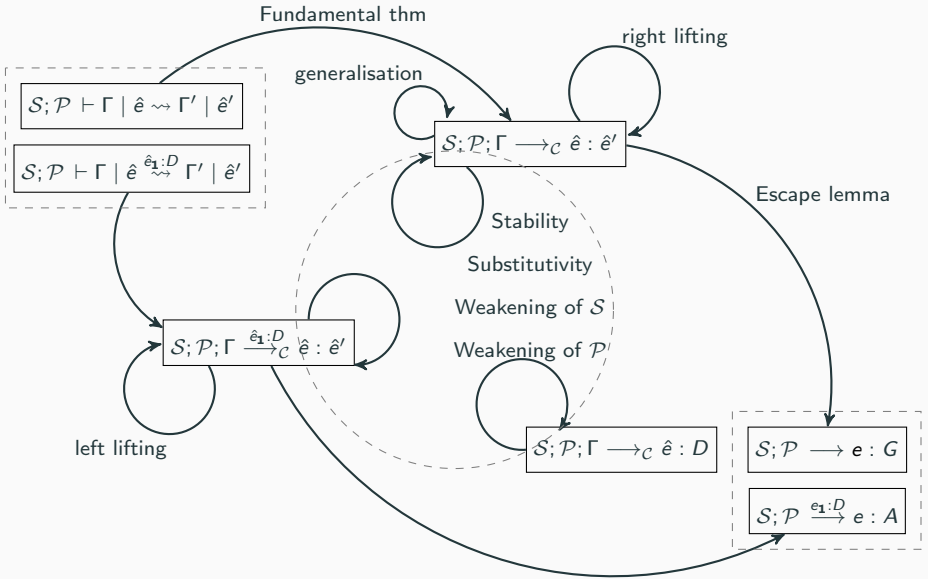
1. Yes, it is
2. proof outline
3. from Small to Big, middle missing. Issue: proof vs mixed, unification vars vs ground
4. we devise logical relation
5. have structural properties
6. have lifting, generalisation

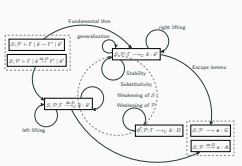




7/10 - 8 min / 30 min total

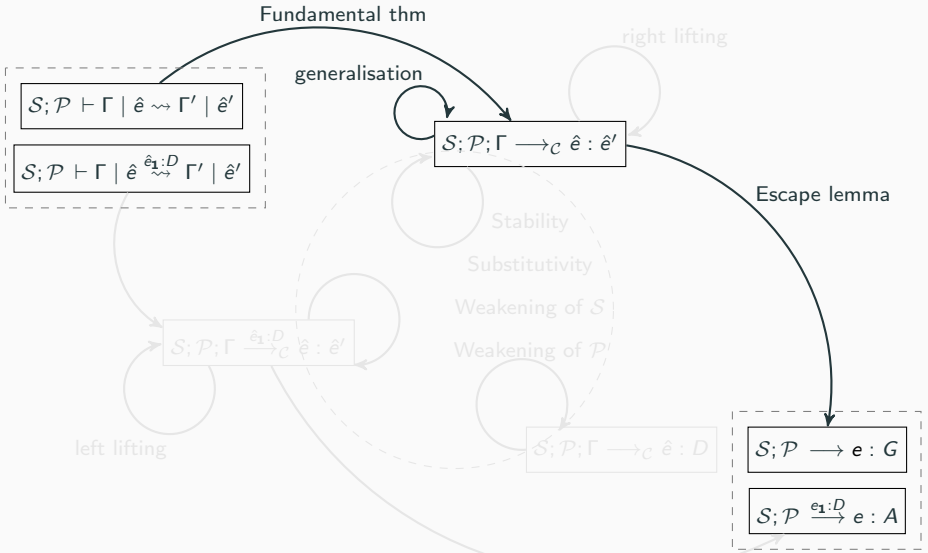
1. Yes, it is
2. proof outline
3. from Small to Big, middle missing. Issue: proof vs mixed, unification vars vs ground
4. we devise logical relation
5. have structural properties
6. have lifting, generalisation
7. fundamental escape - in pairs

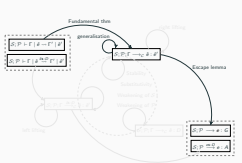




7/10 - 8 min / 30 min total

1. Yes, it is
2. proof outline
3. from Small to Big, middle missing. Issue: proof vs mixed, unification vars vs ground
4. we devise logical relation
5. have structural properties
6. have lifting, generalisation
7. fundamental escape - in pairs
8. let's forget





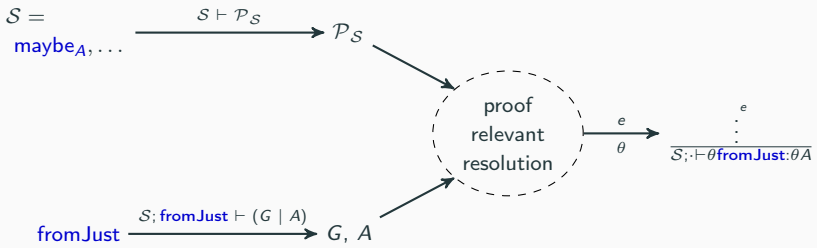
7/10 - 8 min / 30 min total

1. Yes, it is
2. proof outline
3. from Small to Big, middle missing. Issue: proof vs mixed, unification vars vs ground
4. we devise logical relation
5. have structural properties
6. have lifting, generalisation
7. fundamental escape - in pairs
8. let's forget
9. LR makes sense - strong normalisation
10. next - back to examples

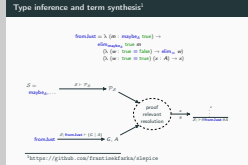
Application to the Examples

Type inference and term synthesis[1]

$$\begin{aligned}\mathsf{fromJust} = \lambda\ (m : \mathsf{maybe}_A\ \mathsf{true}) \to \\ \mathsf{elim}_{\mathsf{maybe}_A}\ \mathsf{true}\ m \\ (\lambda\ (w : \mathsf{true} \equiv \mathsf{false}) \to \mathsf{elim}_{\equiv}\ w) \\ (\lambda\ (w : \mathsf{true} \equiv \mathsf{true})\ (x : A) \to x)\end{aligned}$$



[1] https://github.com/frantisekfarka/slepice



8/10 - 4 min / 34 min total

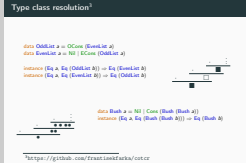
1. proof search for automaton
2. why proof relevant? verfication
3. proof carrying achitecture
4. implementation

Type class resolution[3]

```
data OddList a = OCons (EvenList a)
data EvenList a = Nil | ECons (OddList a)

instance (Eq a, Eq (OddList b)) ⇒ Eq (EvenList b)
instance (Eq a, Eq (EvenList b)) ⇒ Eq (OddList b)
```

```
data Bush a = Nil | Cons (Bush (Bush a))
instance (Eq a, Eq (Bush (Bush b))) ⇒ Eq (Bush b)
```

[3] https://github.com/frantisekfarka/cotcr



9/10 - 4 min / 38 min total

1. posibly nonterminating - coinduction
2. proof system extends
3. semantical considerations - coinductive models
4. proof-relevance helps
5. simple = cycles in HC, more comples - need HH
6. where next

Where to Next

Future work

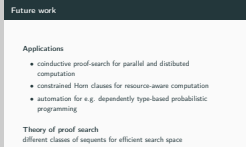
Applications

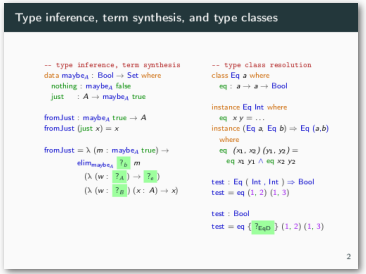
- coinductive proof-search for parallel and distibuted computation
- constrained Horn clauses for resource-aware computation
- automation for e.g. dependently type-based probabilistic programming

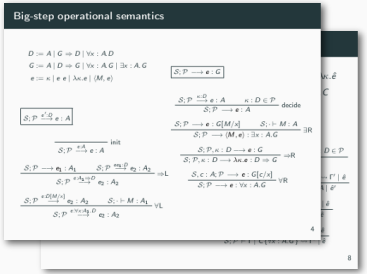
Theory of proof search

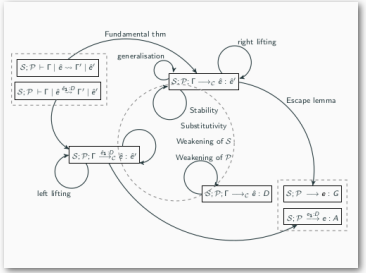
different classes of sequents for efficient search space

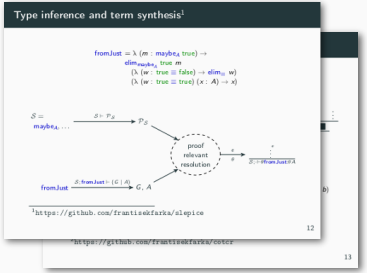
10/10 - 4 min / 42 min total



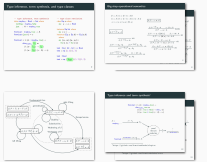








concl - 2 min / 44 min total



Appendix

References i

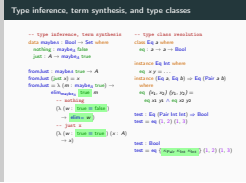
Type inference, term synthesis, and type classes

```
-- type inference, term synthesis
data maybeA : Bool → Set where
  nothing : maybeA false
  just : A → maybeA true

fromJust : maybeA true → A
fromJust (just x) = x
fromJust = λ (m : maybeA true) →
        elim_maybeA true m
          -- nothing
          (λ (w : true ≡ false )
            → elim≡ w )
          -- just x
          (λ (w : true ≡ true ) (x : A)
            → x)
```

```
-- type class resolution
class Eq a where
  eq : a → a → Bool

instance Eq Int where
  eq  x y = ...
instance (Eq a, Eq b) ⇒ Eq (Pair a b)
  where
  eq  (x1 , x2 ) (y1 , y2 ) =
    eq x1 y1 ∧ eq x2 y2

test : Eq (Pair Int Int) ⇒ Bool
test = eq (1, 2) (1, 3)


test : Bool
test = eq {κPair κInt κInt} (1, 2) (1, 3)
```

2019-04-28

└Type inference, term synthesis, and type classes



Type inference, term synthesis, and type classes

```
-- type inference, term synthesis
data maybe_A : Bool → Set where
  nothing : maybe_A false
  just : A → maybe_A true

fromJust : maybe_A true → A
fromJust (just x) = x
fromJust = λ (m : maybe_A true) →
        elim_maybe_A ?_b  m
          -- nothing
          (λ (w : ?_A )
            → ?_e )
          -- just x
          (λ (w : ?_B ) (x : A)
            → x)
```

$$\frac{\Gamma \vdash M : \Pi x : A.B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B[N/x]}\ \Pi\text{-t-Elim}$$

$$\mathtt{elim}_{\equiv_{\mathtt{bool}}} : (\mathtt{tt} \equiv_{\mathtt{bool}} \mathtt{ff}) \to A$$

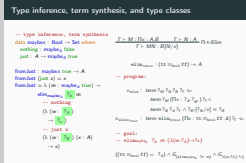
```
-- program:
```

$$\kappa_{\mathtt{elim}} : \mathit{term}\ ?_M\ ?_N\ ?_B\ ?_\Gamma \Leftarrow$$
$$\mathit{term}\ ?_M\ (\Pi x : ?_A.?_{B'})\ ?_\Gamma \wedge$$
$$\mathit{term}\ ?_N\ ?_A\ ?_\Gamma \wedge ?_{B'}[?_N/x] \equiv ?_B$$

$$\kappa_{\mathtt{elim}_{\equiv_{\mathtt{bool}}}} : \mathit{term}\ \mathtt{elim}_{\equiv_{\mathtt{bool}}}\ (\Pi x : \mathtt{tt} \equiv_{\mathtt{bool}} \mathtt{ff}\ .\ A)\ ?_\Gamma \Leftarrow$$

```
-- goal:
-- elim_maybe_A  ?_b  m  (λ(w:?_A)→?_e)
```

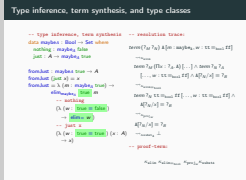
$$((\mathtt{tt} \equiv_{\mathtt{bool}} \mathtt{ff}) = \ ?_A) \wedge G_{(\mathtt{elim}_{\mathtt{maybe_A}}\ \ ?_b\ \ m)} \wedge G_{\lambda(w:?_A).?_e)}$$



Type inference, term synthesis, and type classes

```
-- type inference, term synthesis
data maybeA : Bool → Set where
  nothing : maybeA false
  just : A → maybeA true

fromJust : maybeA true → A
fromJust (just x) = x
fromJust = λ (m : maybeA true) →
        elimmaybeA true m
          -- nothing
          (λ (w : true ≡ false )
            → elim≡ w )
          -- just x
          (λ (w : true ≡ true ) (x : A)
            → x)
```

```
-- resolution trace:

term (?M ?N) A [m : maybeA, w : tt ≡bool ff]

  ⇝κelim

  term ?M (Π x : ?A . A) [...] ∧ term ?N ?A
      [..., w : tt ≡bool ff] ∧ A[?N/x] ≡ ?B

  ⇝κelim≡bool

  term ?N tt ≡bool ff [..., w : tt ≡bool ff] ∧
      A[?N/x] ≡ ?B

  ⇝κprojw

  A[?N/x] ≡ ?B

  ⇝κsubstA ⊥

-- proof-term:

  κelim κelim≡bool κprojw κsubstA
```

Big-step operational semantics

Example

$$\begin{aligned}\mathcal{P} = \kappa_z &: odd(z),\\ \kappa_e &: \forall x : a.odd\ a \Rightarrow even\ (s\ x)\\ \kappa_o &: \forall x : a.even\ a \Rightarrow odd\ (s\ x)\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{S}, c : a; \mathcal{P}, \kappa_x : even\ c \longrightarrow \kappa_x : even\ c}{\mathcal{S}, c : a; \mathcal{P}, \kappa_x : even\ c \longrightarrow \kappa_o\ \kappa_x : odd\ (s\ c)}}{\mathcal{S}, c : a; \mathcal{P}, \kappa_x : even\ c \longrightarrow \kappa_e\ (\kappa_o\ \kappa_x) : even\ (s\ (s\ c))}}{\mathcal{S}, c : a; \mathcal{P} \longrightarrow \lambda\kappa_x.\kappa_e\ (\kappa_o\ \kappa_x) : even\ c \Rightarrow even\ (s\ (s\ c))}}{\mathcal{S}; \mathcal{P} \longrightarrow \lambda\kappa_x.\kappa_e(\kappa_o\kappa_x) : \forall x : a.even\ x \Rightarrow even\ (s\ (s\ x))}$$

Big-step operational semantics
Example

Small-step operational semantics

Example

$$\cdot \mid \forall x : a.even\, x \Rightarrow even\,(s\,(s\,x)) \rightsquigarrow \cdot \mid even\, c \Rightarrow even\,(s\,(s\,c)) \rightsquigarrow$$

$$\cdot \mid \lambda\kappa_x.even\,(s\,(s\,c)) \rightsquigarrow \cdot \mid \lambda\kappa_x.even\,(s\,(s\,c))^{\kappa_e:\forall x:a.odd\, x \Rightarrow even\,(s\,x)} \rightsquigarrow$$

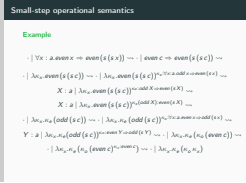
$$X : a \mid \lambda\kappa_x.even\,(s\,(s\,c))^{\kappa_e:odd\, X \Rightarrow even\,(s\,X)} \rightsquigarrow$$

$$X : a \mid \lambda\kappa_x.even\,(s\,(s\,c))^{\kappa_e(odd\, X):even\,(s\,X)} \rightsquigarrow$$

$$\cdot \mid \lambda\kappa_x.\kappa_e\,(odd\,(s\,c)) \rightsquigarrow \cdot \mid \lambda\kappa_x.\kappa_e\,(odd\,(s\,c))^{\kappa_o:\forall x:a.even\, x \Rightarrow odd\,(s\,x)} \rightsquigarrow$$

$$Y : a \mid \lambda\kappa_x.\kappa_e(odd\,(s\,c))^{\kappa_o:even\, Y \Rightarrow odd\,(s\,Y)} \rightsquigarrow \cdot \mid \lambda\kappa_x.\kappa_e\,(\kappa_o\,(even\, c)) \rightsquigarrow$$

$$\cdot \mid \lambda\kappa_x.\kappa_e\,(\kappa_o\,(even\, c)^{\kappa_x:even\, c}) \rightsquigarrow \cdot \mid \lambda\kappa_x.\kappa_e\,(\kappa_o\,\kappa_x)$$



Logical relation

$\boxed{\mathcal{S};\mathcal{P};\Gamma \longrightarrow_{\mathcal{C}} \hat{e} : \hat{e}'}$

$$\frac{\mathcal{S};\mathcal{P};\Gamma \xrightarrow[\mathcal{C}]{\hat{e}':D} \hat{e} : A \qquad \mathcal{S};\mathcal{P};\Gamma \longrightarrow_{\mathcal{C}} \hat{e}' : D}{\mathcal{S};\mathcal{P};\Gamma \longrightarrow_{\mathcal{C}} \hat{e} : A}$$

$$\frac{\mathcal{S};\mathcal{P};\Gamma \longrightarrow_{\mathcal{C}} \hat{e} : G[M/x] \qquad \mathcal{S};\Gamma \vdash M : A}{\mathcal{S};\mathcal{P};\Gamma \longrightarrow_{\mathcal{C}} \langle M, \hat{e}\rangle : \exists x : A.G}$$

$$\frac{\mathcal{S};\mathcal{P},\kappa : D;\Gamma \longrightarrow_{\mathcal{C}} \hat{e} : G}{\mathcal{S};\mathcal{P};\Gamma \longrightarrow_{\mathcal{C}} \lambda\kappa.\hat{e} : D \Rightarrow G}$$

$$\frac{\mathcal{S},c : A;\mathcal{P};\Gamma \longrightarrow_{\mathcal{C}} \hat{e}[c/x] : G[c/x]}{\mathcal{S};\mathcal{P};\Gamma \longrightarrow_{\mathcal{C}} \hat{e} : \forall x : A.G}$$

$$\frac{\mathcal{S} \vdash \mathcal{P} \qquad \mathcal{S} \vdash \Gamma}{\mathcal{S};\mathcal{P};\Gamma \longrightarrow_{\mathcal{C}} A : A}$$

$$\frac{\mathcal{S};\mathcal{P};\Gamma \longrightarrow_{\mathcal{C}} \hat{e}_1 : \hat{e}_2}{\mathcal{S};\mathcal{P};\Gamma \longrightarrow_{\mathcal{C}} (\theta\hat{e})\,\hat{e}_1 : \hat{e}\,\hat{e}_2}$$

$$\frac{\mathcal{S};\mathcal{P};\Gamma \longrightarrow_{\mathcal{C}} \hat{e}_1 : \hat{e}_2}{\mathcal{S};\mathcal{P};\Gamma \longrightarrow_{\mathcal{C}} \langle \theta M, \hat{e}_1\rangle : \langle M, \hat{e}_2\rangle}$$

$$\frac{\mathcal{S};\mathcal{P};\Gamma \longrightarrow_{\mathcal{C}} \hat{e}_1 : \hat{e}_2}{\mathcal{S};\mathcal{P};\Gamma \longrightarrow_{\mathcal{C}} \lambda\kappa.\hat{e}_1 : \lambda\kappa.\hat{e}_2}$$

$\boxed{\mathcal{S};\mathcal{P};\Gamma \xrightarrow[\mathcal{C}]{\hat{e}:D} \hat{e} : \hat{e}'}$

$$\frac{}{\mathcal{S};\mathcal{P};\Gamma \xrightarrow[\mathcal{C}]{\hat{e}:A} \hat{e} : A}$$

$$\frac{\mathcal{S};\mathcal{P};\Gamma \longrightarrow_{\mathcal{C}} \hat{e}_1 : A_1 \qquad \mathcal{S};\mathcal{P};\Gamma \xrightarrow[\mathcal{C}]{\hat{e}\,\hat{e}_1:D} \hat{e}_2 : A_2}{\mathcal{S};\mathcal{P};\Gamma \xrightarrow[\mathcal{C}]{\hat{e}:A_1 \Rightarrow D} \hat{e}_2 : A_2}$$

$$\frac{\mathcal{S};\mathcal{P};\Gamma \xrightarrow[\mathcal{C}]{\hat{e}:D[M/x]} \hat{e}_2 : A_2 \qquad \mathcal{S};\Gamma \vdash M : A_1}{\mathcal{S};\mathcal{P};\Gamma \xrightarrow[\mathcal{C}]{\hat{e}:\forall x:A_1.D} \hat{e}_2 : A_2}$$

$\boxed{\mathcal{S};\mathcal{P};\Gamma \longrightarrow_{\mathcal{C}} \hat{e} : D}$

$$\frac{\mathcal{S} \vdash \mathcal{P} \qquad \kappa : D \in \mathcal{P} \qquad \mathcal{S} \vdash \Gamma}{\mathcal{S};\mathcal{P};\Gamma \longrightarrow_{\mathcal{C}} \kappa : D}$$

$$\frac{\mathcal{S};\mathcal{P};\Gamma \longrightarrow_{\mathcal{C}} \hat{e} : A \Rightarrow D \qquad \mathcal{S};\mathcal{P};\Gamma \longrightarrow_{\mathcal{C}} \hat{e}' : A}{\mathcal{S};\mathcal{P};\Gamma \longrightarrow_{\mathcal{C}} \hat{e}\,\hat{e}' : D}$$

$$\frac{\mathcal{S};\mathcal{P};\Gamma \longrightarrow_{\mathcal{C}} \hat{e} : \forall x : A.D \qquad \mathcal{S};\Gamma \vdash M : A}{\mathcal{S};\mathcal{P};\Gamma \longrightarrow_{\mathcal{C}} \hat{e} : D[M/x]}$$

Type class resolution - Pair

Example

$\mathcal{P}_{Pair} =$

$$\begin{array}{lll} \kappa_1 : & \mathtt{eq}(x),\ \mathtt{eq}(y) & \Rightarrow \mathtt{eq}(\mathtt{pair}(x, y)) \\ \kappa_2 : & & \Rightarrow \mathtt{eq}(\mathtt{int}) \end{array}$$

$$\dfrac{\dfrac{}{\mathcal{P}_{Pair} \longrightarrow \kappa_2 : \mathtt{eq}(\mathtt{int})}\ \text{Lp-m} \qquad \dfrac{}{\mathcal{P}_{Pair} \longrightarrow \kappa_2 : \mathtt{eq}(\mathtt{int})}\ \text{Lp-m}}{\mathcal{P}_{Pair} \longrightarrow \kappa_1\kappa_2\kappa_2 : \mathtt{eq}(\mathtt{pair}(\mathtt{int}, \mathtt{int}))}\ \text{Lp-m}$$

Type class resolution - Pair
Example
$\mathcal{P}_{Pair} =$
$\kappa_1 : \mathtt{eq}(x),\ \mathtt{eq}(y) \Rightarrow \mathtt{eq}(\mathtt{pair}(x, y))$
$\kappa_2 : \Rightarrow \mathtt{eq}(\mathtt{int})$
$\dfrac{\dfrac{}{\mathcal{P}_{Pair} \longrightarrow \kappa_2 : \mathtt{eq}(\mathtt{int})}\ \text{Lp-m} \quad \dfrac{}{\mathcal{P}_{Pair} \longrightarrow \kappa_2 : \mathtt{eq}(\mathtt{int})}\ \text{Lp-m}}{\mathcal{P}_{Pair} \longrightarrow \kappa_1\kappa_2\kappa_2 : \mathtt{eq}(\mathtt{pair}(\mathtt{int}, \mathtt{int}))}\ \text{Lp-m}$

Type class resolution - Bush

Example

$\mathcal{P}_{Bush} =$

$$\kappa_1 : \quad \Rightarrow \mathtt{eq}(\mathtt{int})$$

$$\kappa_2 : \ \mathtt{eq}(x),\ \mathtt{eq}(\mathtt{bush}(\mathtt{bush}(x))) \Rightarrow \mathtt{eq}(\mathtt{bush}(x))$$

$$\dfrac{\dfrac{}{\mathcal{P}_{Bush} \longrightarrow \kappa_1 : \mathtt{eq}(\mathtt{int})} \quad \dfrac{\dfrac{\dfrac{\vdots}{\mathcal{P}_{Bush}, (\alpha : \mathtt{eq}(x) \Rightarrow \mathtt{eq}(\mathtt{bush}(x))), (\beta : \ \Rightarrow \mathtt{eq}(x)) \longrightarrow \kappa_2\beta(\alpha(\alpha\beta)) : \mathtt{eq}(\mathtt{bush}(x))}}{\mathcal{P}_{Bush}, (\alpha : \_) \longrightarrow \lambda\beta.\kappa_2\beta(\alpha(\alpha\beta)) : \mathtt{eq}(x) \Rightarrow \mathtt{eq}(\mathtt{bush}(x))}\,\text{Lam}}{\mathcal{P}_{Bush} \longrightarrow \nu\alpha.\lambda\beta.\kappa_2\beta(\alpha(\alpha\beta)) : \mathtt{eq}(x) \Rightarrow \mathtt{eq}(\mathtt{bush}(x))}\,\text{Nu}}{\mathcal{P}_{Bush} \longrightarrow (\nu\alpha.\lambda\beta.\kappa_2\beta(\alpha(\alpha\beta)))\kappa_1 : \mathtt{eq}(\mathtt{bush}(\mathtt{int}))}$$